

## **Blockchain Lab**

### **Experiment-2**

Sanket More  
D20A 35

**Aim:** Create a Blockchain using Python

#### **Theory:-**

#### **Introduction to Blockchain Technology**

Blockchain is a **decentralized digital ledger system** designed to securely record transactions across a distributed network of computers (called *nodes*). Instead of storing data in a single central database, blockchain distributes identical copies of the ledger across many participants.

Transactions are grouped into **blocks**, and each block is cryptographically linked to the previous one using hash functions, forming a continuous **chain of blocks**. This design ensures that once information is recorded, it becomes extremely difficult to alter or delete, thereby providing a **permanent, transparent, and tamper-resistant record**.

The core characteristics of blockchain are:

#### **1. Decentralization**

No single authority controls the network. Validation and maintenance are performed collectively by multiple nodes.

#### **2. Transparency**

All verified transactions are visible to network participants, promoting trust and accountability.

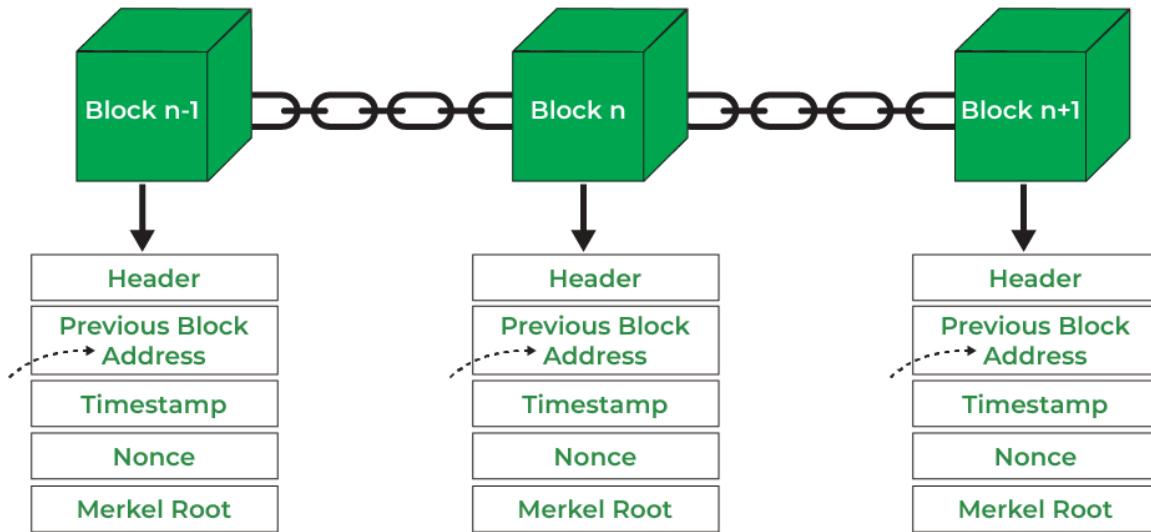
#### **3. Immutability**

After a block is added, modifying its data would require changing all subsequent blocks and gaining consensus from most of the network—making fraud practically infeasible.

## What Is a Block?

A **block** is the fundamental data unit of a blockchain. It functions like a page in a digital record book, storing a collection of validated transactions along with essential metadata. Each block is securely connected to the previous block, forming an unbroken chain.

Blocks help organize and preserve transaction history while maintaining integrity through cryptographic techniques.



## Main Components of a Block

### 1. Block Header

The block header uniquely identifies a block within the blockchain. It contains critical metadata and is repeatedly hashed during mining. The header typically includes:

- Previous block hash
- Merkle root
- Timestamp
- Nonce

## 2. Previous Block Hash (Block Address)

This field stores the hash of the preceding block, linking blocks together and ensuring continuity of the chain.

## 3. Timestamp

Indicates the exact date and time when the block was created. It helps maintain chronological order and validates data authenticity.

## 4. Nonce

A “number used once” that miners modify during the mining process to produce a valid hash that satisfies the network’s difficulty level.

## 5. Merkle Root

A single hash representing all transactions within the block. It is generated using a Merkle Tree and enables efficient verification of transaction inclusion.

## Example of a Block

**Block Number:** 105678

**Block Header:**

- Previous Block Hash:

00000000000000000000a3f2c9e7b1d45c8fa9a1e7b6d9c2f0a4e3b1c8d9  
f

- Merkle Root:

4f3c2a1b9e8d7c6b5a4f3e2d1c9b8a7f6e5d4c3b2a1f9e8d7c6b5a4

- Timestamp:

2026-01-29 14:32:10 UTC

- Nonce:

845932

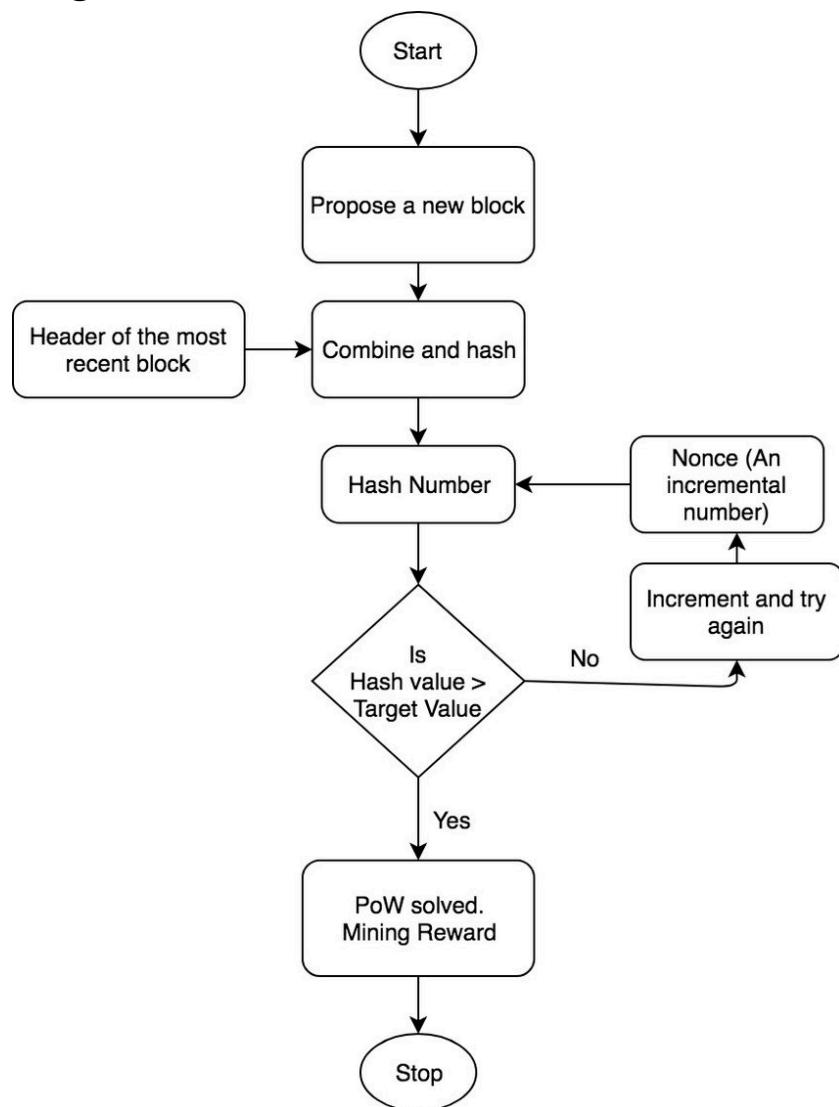
- Block Hash:

`0000000000000000000000005d6e8f3c2a9b1e7d4f6c8a2e9b5d3f1c7...`

### Transactions:

1. Alice → Bob : 0.5 BTC
2. Charlie → Dave : 1.2 BTC
3. Eve → Frank : 0.3 BTC

### Process of Mining



Mining is the process by which new blocks are created and added to the blockchain using a mechanism called **Proof of Work (PoW)**.

## Step 1: Collect Transactions

All pending transactions from the network are gathered into a new block.

*Example: Alice sends 50 coins to Bob.*

## Step 2: Create Block Header

The miner prepares a block header containing:

- Previous block hash
- Transaction data (Merkle root)
- Timestamp
- Nonce (initially set to zero)

## Step 3: Proof of Work

Miners repeatedly change the nonce and compute the hash until it satisfies the network's difficulty condition.

Example:

**SHA-256(Block Data +Nonce) → 0000ab34cd...**

The required number of leading zeros defines the **difficulty level**. This process demands significant computational power, hence the name *Proof of Work*.

## Step 4: Validation and Broadcasting

Once a valid hash is discovered, the block is broadcast to the network. Other nodes verify:

1. Hash correctness
2. Nonce validity

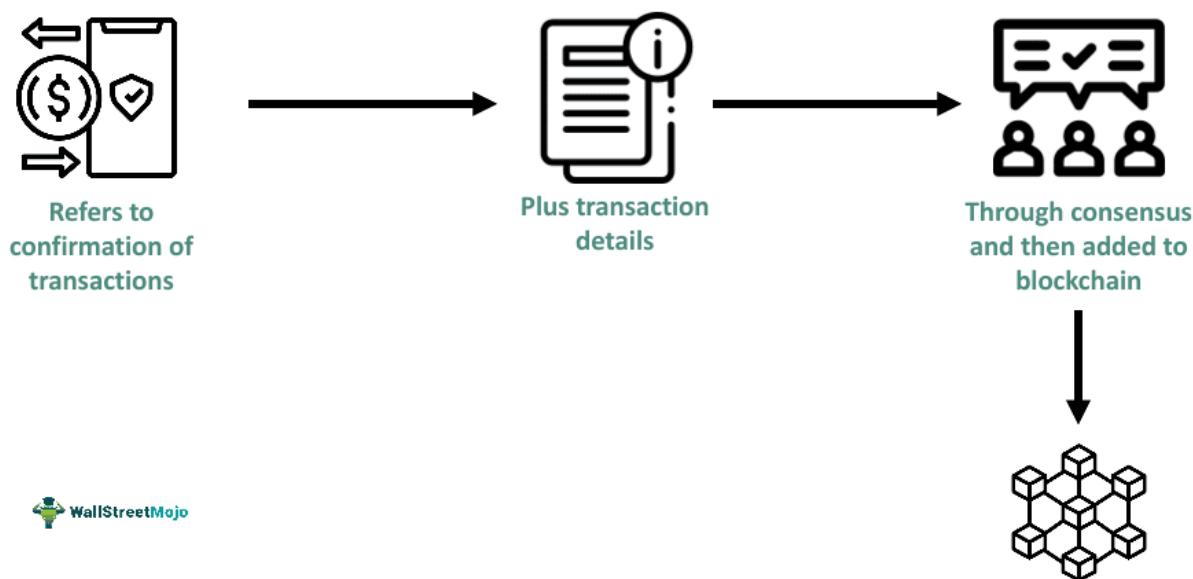
- Link to the previous block

## Step 5: Add Block to Blockchain

After successful verification, the block is permanently appended to the blockchain, and the miner receives a cryptocurrency reward.

## How to Check the Validity of a Block

### Meaning Of Block Validation



To ensure network security, each block undergoes multiple validation checks:

#### 1. Verify Block Hash

Recalculate the hash from the block header and compare it with the stored hash.

#### 2. Check Proof of Work

Confirm that the hash satisfies the required difficulty (leading zeros).

Example:

Required: **0000xxxx**

Found: **0000f9a2**

### **3. Validate Previous Block Hash**

Ensure the stored previous hash matches the hash of the last accepted block.

### **4. Verify Transactions**

- Confirm transaction authenticity
- Prevent double spending
- Validate digital signatures
- Ensure sufficient account balances

### **5. Verify Merkle Root**

Recompute the Merkle root from all transactions and compare it with the value in the block header.

### **6. Check Timestamp**

Ensure the timestamp is valid and not set in the future.

### **7. Check Block Size and Format**

Verify that block size and structure comply with protocol rules.

### **8. Validate Genesis Block**

The first block (Genesis Block) is predefined:

- Previous hash =  $\emptyset$
- Must remain unchanged

## **Final Result**

- If all verification steps pass → The block is accepted as valid.
- If any step fails → The block is rejected by the network.

## CODE:-

```
import datetime
import hashlib
import json
from flask import Flask, jsonify

# -----
# Blockchain Class Definition
# -----


class Blockchain:

    def __init__(self):
        self.chain = []
        # Create Genesis Block
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash
        }
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        while True:
            hash_operation = hashlib.sha256(
                str(new_proof**2 - previous_proof**2).encode()
            ).hexdigest()

            if hash_operation[:4] == '0000':
                return new_proof


```

```

new_proof += 1

def hash(self, block):
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()

def is_chain_valid(self, chain):
    previous_block = chain[0]
    index = 1

    while index < len(chain):
        block = chain[index]

        # Check previous hash
        if block['previous_hash'] != self.hash(previous_block):
            return False

        # Check Proof of Work
        hash_operation = hashlib.sha256(
            str(block['proof']**2 -
                previous_block['proof']**2).encode()
            ).hexdigest()

        if hash_operation[:4] != '0000':
            return False

        previous_block = block
        index += 1

    return True

# -----
# Flask Application
# -----


app = Flask(__name__)
blockchain = Blockchain()

# Mine a new block

```

```

@app.route('/mine', methods=['GET'])
def mine():
    previous_block = blockchain.get_previous_block()
    proof = blockchain.proof_of_work(previous_block['proof'])
    previous_hash = blockchain.hash(previous_block)
    block = blockchain.create_block(proof, previous_hash)

    return jsonify({
        'student_name': 'Sanket More',
        'roll_no': 35,
        'message': 'Congratulations Sanket More (Roll No 35)! You have
successfully mined a block.',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }), 200


# Display full blockchain
@app.route('/chain', methods=['GET'])
def chain():
    return jsonify({
        'student_name': 'Sanket More',
        'roll_no': 35,
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }), 200


# Validate blockchain
@app.route('/validate', methods=['GET'])
def validate():
    if blockchain.is_chain_valid(blockchain.chain):
        return jsonify({
            'student_name': 'Sanket More',
            'roll_no': 35,
            'message': 'Blockchain is valid. Verified by Sanket More (Roll
No 35).'
        }), 200

```

```

    return jsonify({
        'student_name': 'Sanket More',
        'roll_no': 35,
        'message': 'Blockchain is NOT valid. Checked by Sanket More (Roll
No 35).'
    }), 200

# Run Server
app.run(host='0.0.0.0', port=5000)

```

#### OUTPUT:-

PS C:\Users\Sanket More\Desktop\Blockchain> python app.py

- \* Serving Flask app 'app'
- \* Debug mode: off

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

- \* Running on all addresses (0.0.0.0)
- \* Running on http://127.0.0.1:5000
- \* Running on http://192.168.29.252:5000

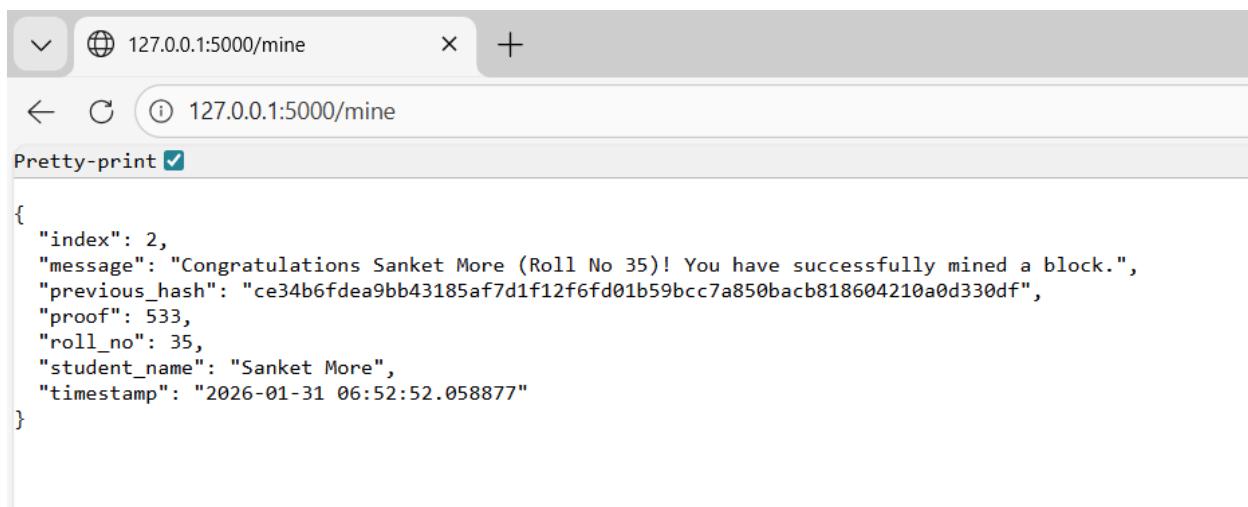
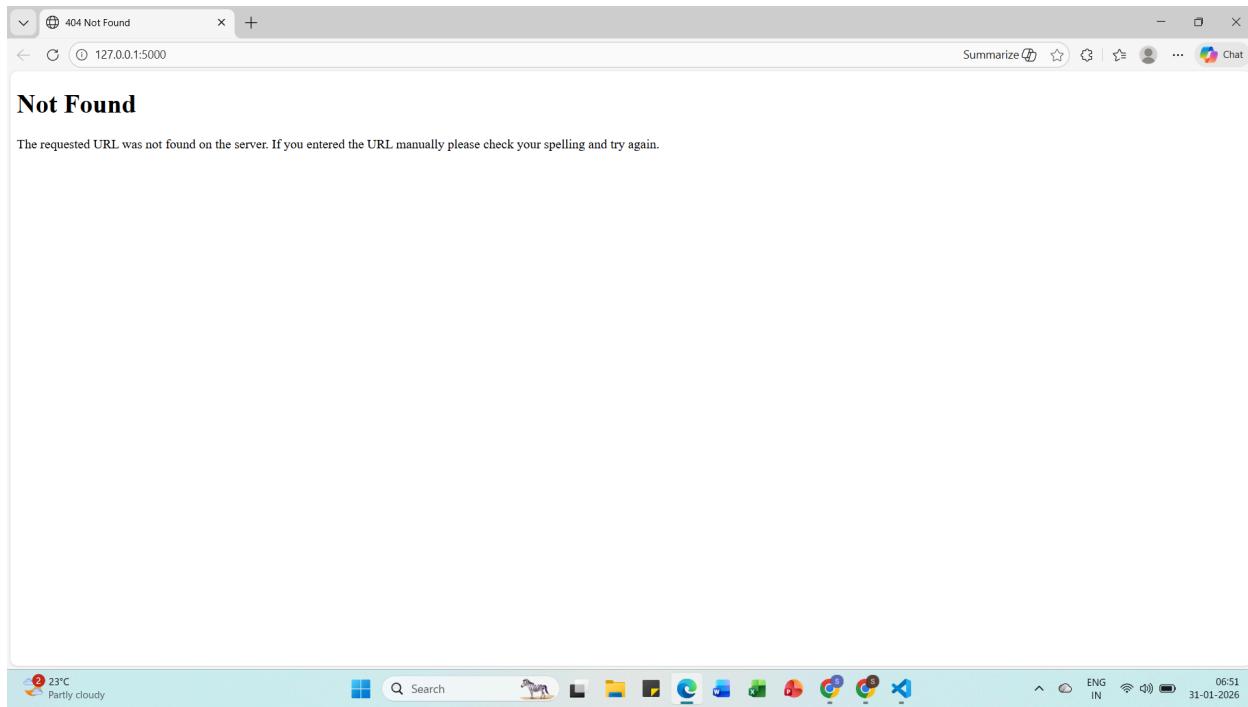
Press CTRL+C to quit

```

127.0.0.1 - - [31/Jan/2026 06:54:54] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [31/Jan/2026 06:54:54] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [31/Jan/2026 06:55:01] "GET /mine HTTP/1.1" 200 -
127.0.0.1 - - [31/Jan/2026 06:55:21] "GET /mine HTTP/1.1" 200 -
127.0.0.1 - - [31/Jan/2026 06:55:49] "GET /chain HTTP/1.1" 200 -
127.0.0.1 - - [31/Jan/2026 06:56:12] "GET /validate HTTP/1.1" 200 -

```

PS C:\Users\Sanket More\Desktop\Blockchain>



127.0.0.1:5000/chain

← ⌂ 127.0.0.1:5000/chain

Pretty-print

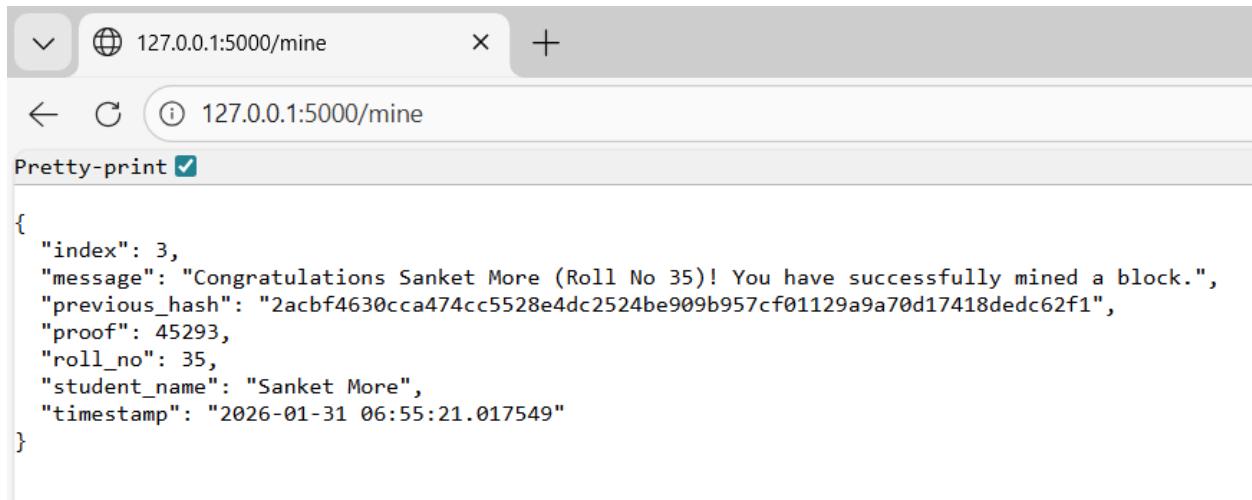
```
{  
  "chain": [  
    {  
      "index": 1,  
      "previous_hash": "0",  
      "proof": 1,  
      "timestamp": "2026-01-31 06:51:36.248894"  
    },  
    {  
      "index": 2,  
      "previous_hash": "ce34b6fdea9bb43185af7d1f12f6fd01b59bcc7a850bacb818604210a0d330df",  
      "proof": 533,  
      "timestamp": "2026-01-31 06:52:52.058877"  
    }  
,  
    "length": 2,  
    "roll_no": 35,  
    "student_name": "Sanket More"  
}
```

127.0.0.1:5000/validate

← ⌂ 127.0.0.1:5000/validate

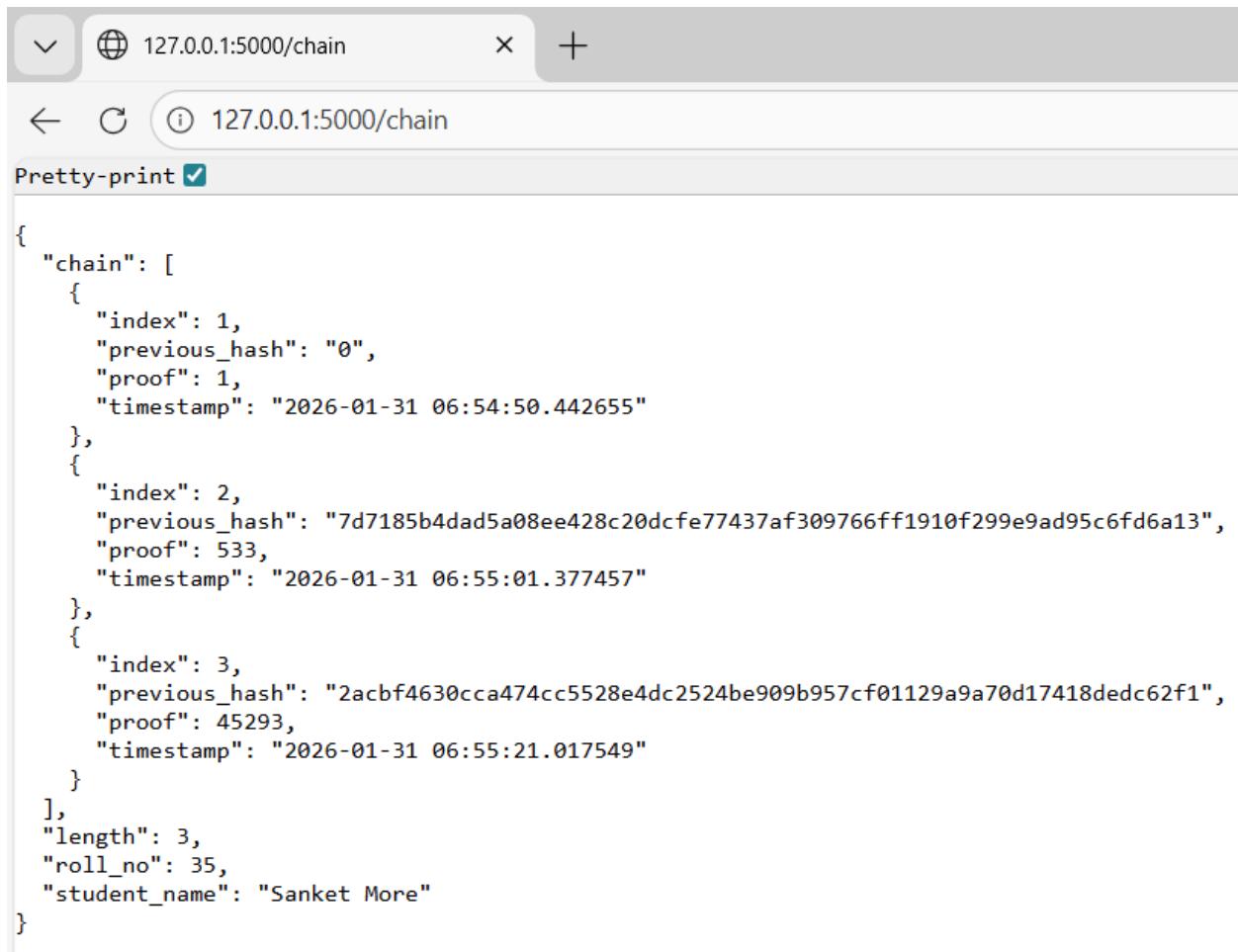
Pretty-print

```
{  
  "message": "Blockchain is valid. Verified by Sanket More (Roll No 35).",  
  "roll_no": 35,  
  "student_name": "Sanket More"  
}
```



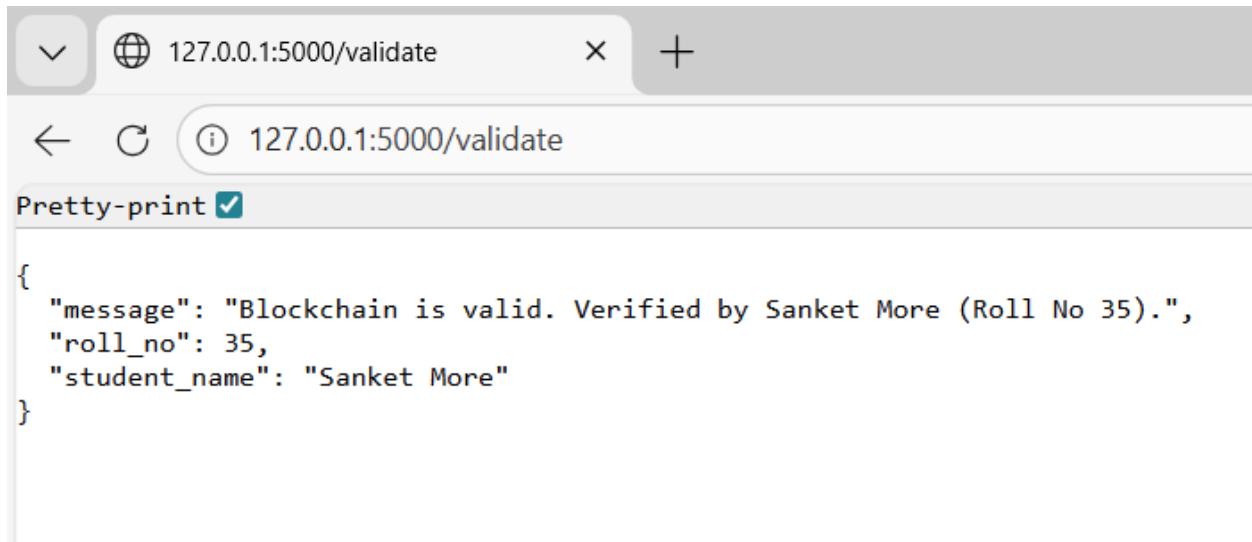
A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:5000/mine`. The page content is a JSON object representing a mined block:

```
{  
  "index": 3,  
  "message": "Congratulations Sanket More (Roll No 35)! You have successfully mined a block.",  
  "previous_hash": "2acbf4630cca474cc5528e4dc2524be909b957cf01129a9a70d17418dedc62f1",  
  "proof": 45293,  
  "roll_no": 35,  
  "student_name": "Sanket More",  
  "timestamp": "2026-01-31 06:55:21.017549"  
}
```



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:5000/chain`. The page content is a JSON object representing a blockchain chain:

```
{  
  "chain": [  
    {  
      "index": 1,  
      "previous_hash": "0",  
      "proof": 1,  
      "timestamp": "2026-01-31 06:54:50.442655"  
    },  
    {  
      "index": 2,  
      "previous_hash": "7d7185b4dad5a08ee428c20dcfe77437af309766ff1910f299e9ad95c6fd6a13",  
      "proof": 533,  
      "timestamp": "2026-01-31 06:55:01.377457"  
    },  
    {  
      "index": 3,  
      "previous_hash": "2acbf4630cca474cc5528e4dc2524be909b957cf01129a9a70d17418dedc62f1",  
      "proof": 45293,  
      "timestamp": "2026-01-31 06:55:21.017549"  
    }  
  "length": 3,  
  "roll_no": 35,  
  "student_name": "Sanket More"  
}
```



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:5000/validate`. Below the address bar, there is a "Pretty-print" checkbox which is checked. The main content area displays the following JSON response:

```
{  
  "message": "Blockchain is valid. Verified by Sanket More (Roll No 35).",  
  "roll_no": 35,  
  "student_name": "Sanket More"  
}
```

## CONCLUSION:-

The objective of implementing a blockchain using Python was successfully accomplished. Through this experiment, the fundamental components of blockchain technology—including block creation, transaction storage, timestamping, cryptographic hashing, nonce generation, and linking of consecutive blocks—were effectively demonstrated. The Proof-of-Work (PoW) mechanism was implemented to simulate the mining process, ensuring data integrity and network security. Each newly generated block was validated by verifying its hash value, reference to the previous block, and compliance with the difficulty condition. This practical implementation provided valuable insight into how blockchain achieves decentralization, transparency, immutability, and secure transaction recording. Therefore, the core working principles of blockchain technology were successfully studied, implemented, and verified using Python.